**CSE 315 DIGITAL LOGIC DESIGN TERM PROJECT**



**150117057 Rümeysa ÖZTÜRK**

**150116010 Şükriye SOYER**

**150117034 Berra MERCAN**

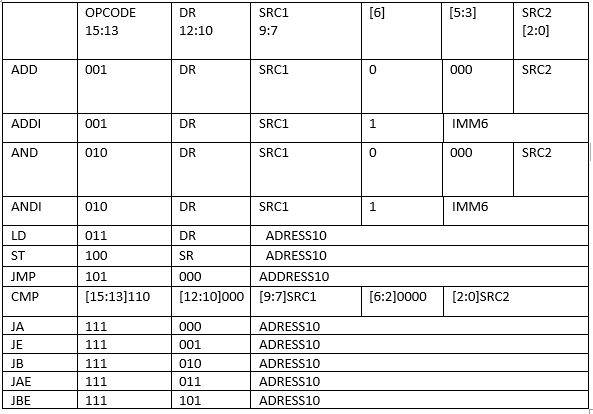
**ASSEMBLER PART**

Our instructions are :

AND, ADD, LD, ST, ANDI, ADDI, CMP, JUMP, JE, JA, JB, JBE, JAE

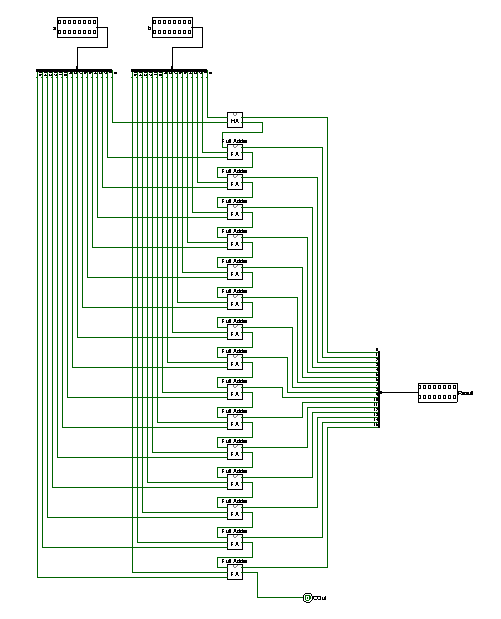
We have 16 bits for our processor. There are 8 registers. So we have 3 bits for representing registers. Also, we used 3 bits for representing opcodes. For immediate values we used 6 bits and for addresses 10 bits. We used same opcodes for ADD and ADDI, AND and ANDI. We distinguish these instructions by looking the 6th bit. We used the same bits for JE, JA, JB, JBE, JAE. We distinguish these instructions by controlling the bits between 12th and 10th. We write a Java code for converting these processors to hexadecimal value.

**ISA:**



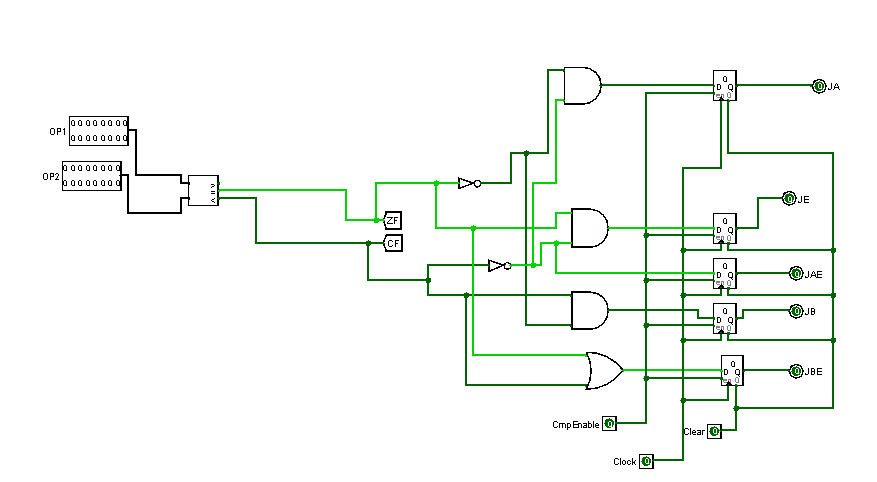
**LOGISIM COMPONENT DESIGN**

**16-Bit Adder**



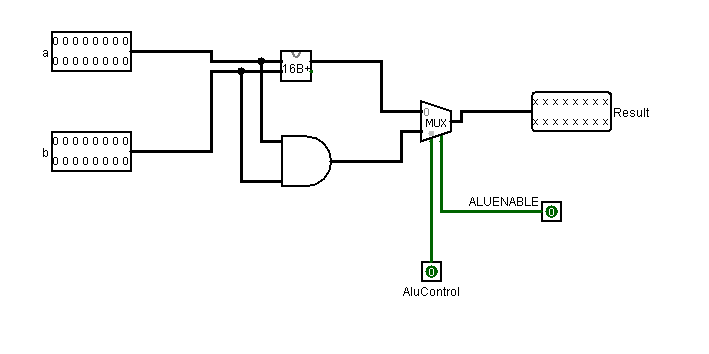
For adding 16-bit data we used half adder and full adder that we constructed ourselves.

**16 Bit Comparator**



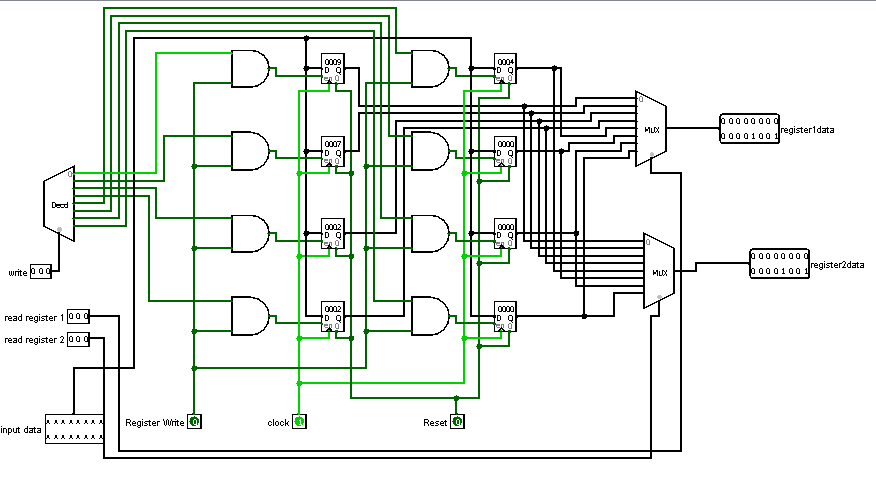
We used logisim’s own comparator for comparating 16-bit data. We set the condition codes, ZF and CF, according to the result that comes from 16-bit comparator. Then, according to ZF and CF we constructed signals, JA,JE,JB,JAE,JBE.

**Arithmetic Logic Unit**



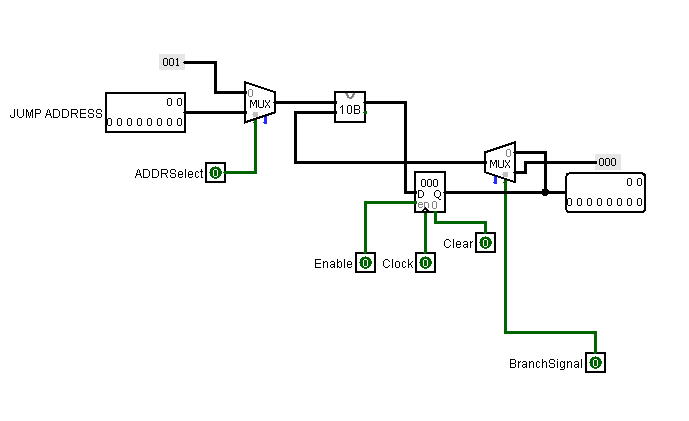
For constructing ALU we used instructions ADD,AND,ADDI,ANDI that we did ourselves. We get inputs ALUENABLE and AluControl. Then, we used them in control unit as signals. AluControl determines instructions, ADD or AND.

**Register File**



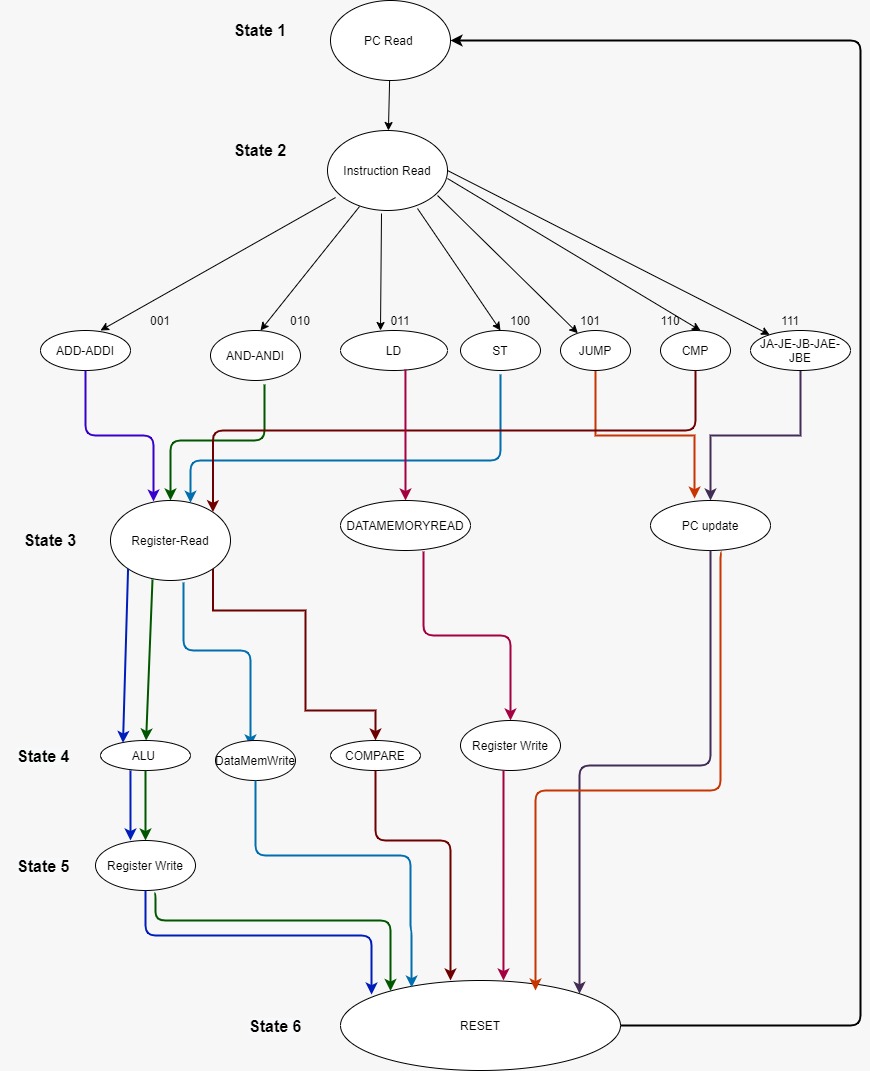
In register file, we used 8 registers. We used two decoders for parallel reading. We used 3 signals, RegisterWrite,Clock and Reset. We send them to control unit and main.

**JUMP**

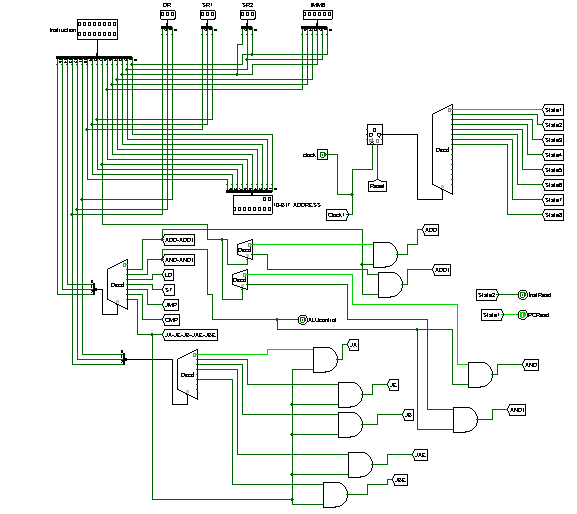


We constructed our own JUMP. We are increasing PC by 1. We used ADDRSelect signal to choose whether PC is going to increased by 1 or by the JUMP ADDRESS. We used BranchSignal in order to avoid oscillation.

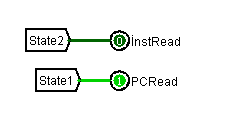
**Finite State Machine**



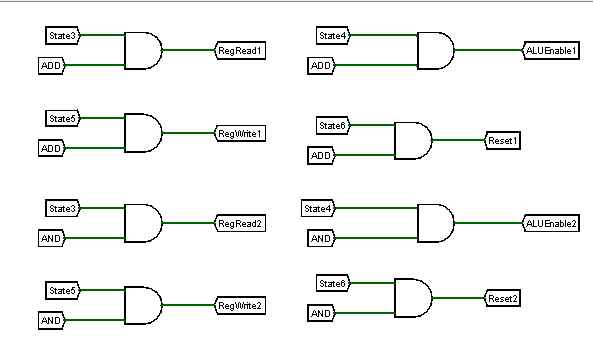
**LOGISIM DESIGN WITH CONTROL UNIT**

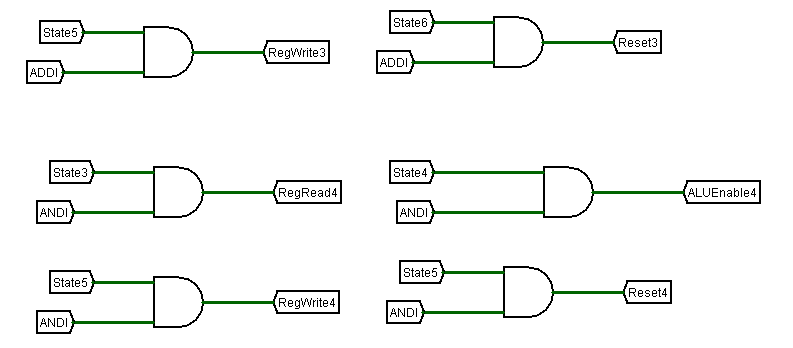


In the control unit we design this component to decoding instructions.

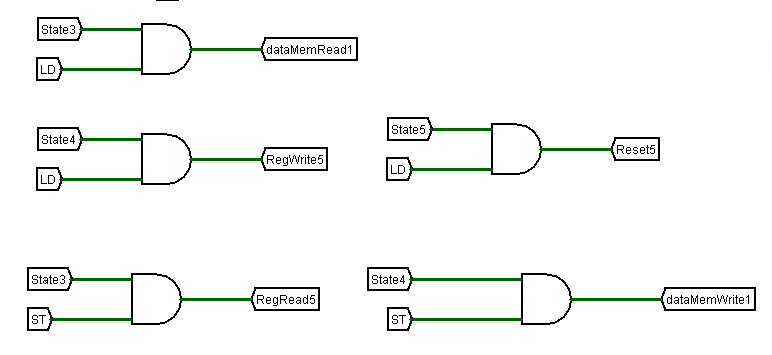


State1 and State2 are same for all instructions.We constructed PCread and InstRead signal.

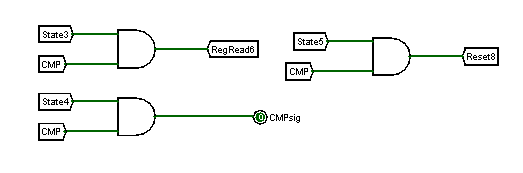
 We constructed states for ADD and AND instructions as shown in the above picture.



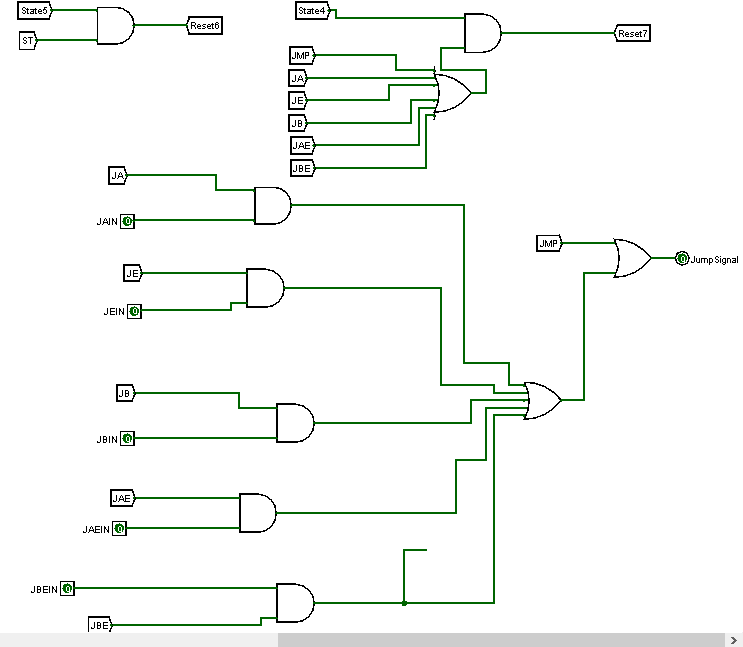
We constructed states for ADDI and ANDI instructions as shown in the above picture.



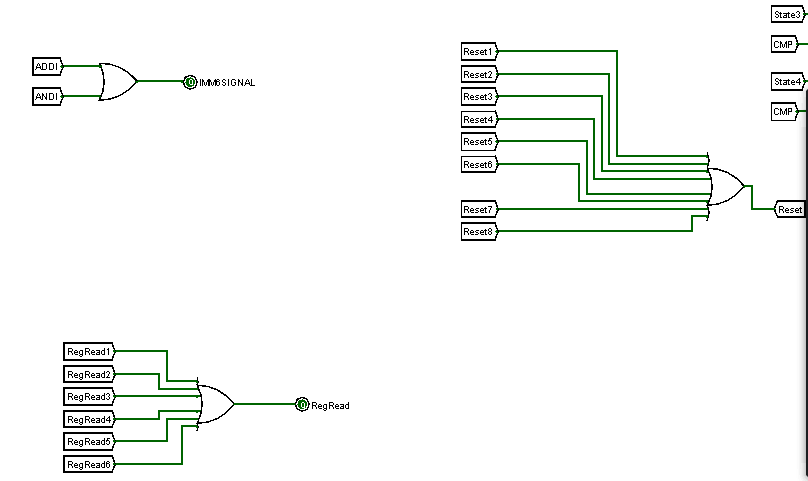
We constructed states for ST and LD instructions as shown in the above picture

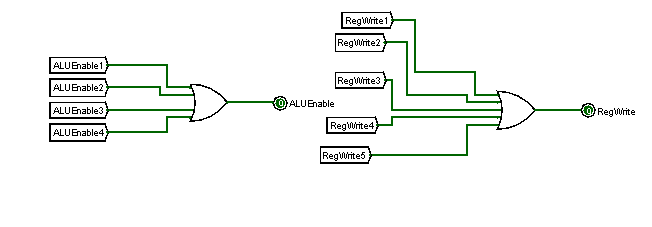


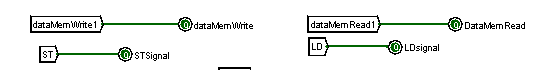
We constructed states for CMP instruction as shown in the above picture



We constructed states for JUMP,JA,JE,JB,JAE and JBE instructions as shown in the above picture

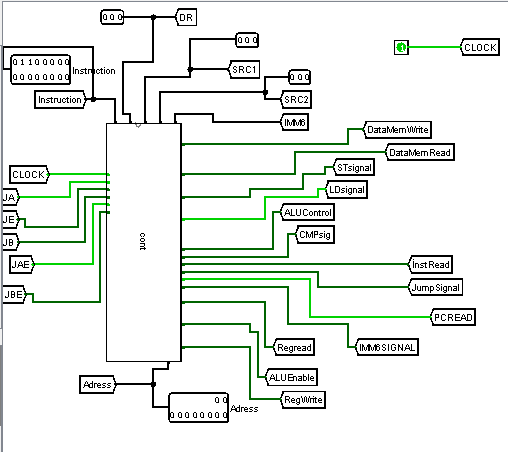




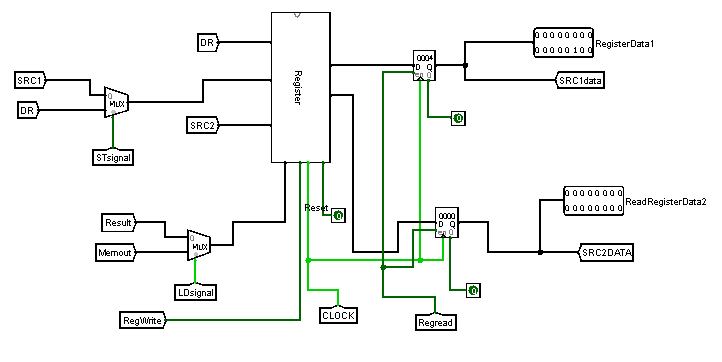


We constructed IMMSIGNAL, Reset, RegRead, RegRead, ALUEnable, RegWrite, dataMemWrite, dataMemRead signals for some states.

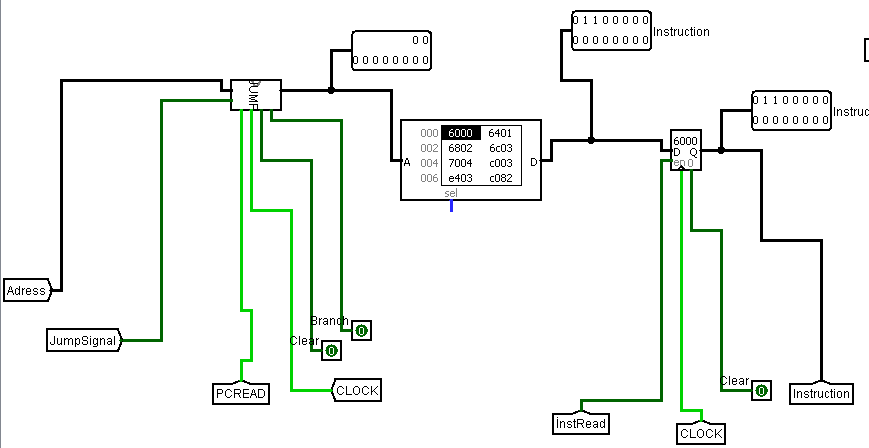
**MAIN PART**



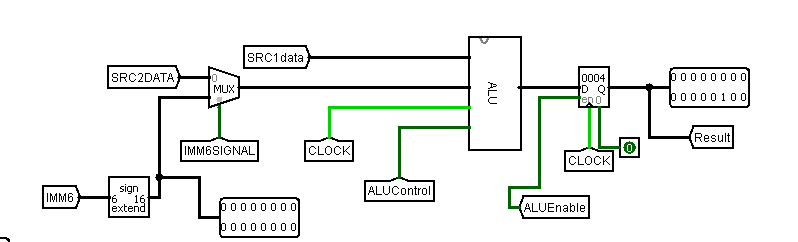
In our main part, we used control unit as shown above. Here we get the signals according to instructions that we read. By clicking on Clock, the program passes to next state.



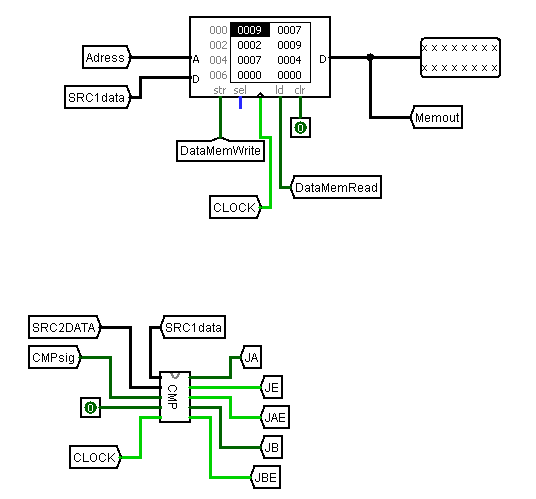
Here, we used our register file for uploading and reading registers’ values.



Here we upload our input file to instruction memory. In JUMP part, we get PC value.



Here, we decided whether we are going to use IMM value or the register’s value.



Here, we read values from memory and wrote them to register.

Also, we wrote the values from register to memory according to 10-Bit Address.

In compare part, we compared SRC2DATA and SRC1data and according to ZF and CF, we decided which instruction will be selected from JA,JE,JB,JAE,JBE.